Wrocław, 6.04.2020 r

Michał Ryśkiewicz, 241383
Termin: WT/TN/7:30

Ocena:

Sprawozdanie z laboratorium nr 3 z przedmiotu

„Organizacja i Architektura Komputerów”

Rok akademicki 2019/2020, kierunek: INF

Prowadzący:

Mgr inż. Tomasz Serafin

1. Cel ćwiczenia

Zadaniem realizowanym podczas zajęć było opracowanie kalkulatora operującego na liczbach typu floating point pojedynczej i podwójnej precyzji. Operacje arytmetyczne dostępne w kalkulatorze to: dodawanie, odejmowanie, mnożenie i dzielenie.

Głównym celem tych ćwiczeń było m. in. zapoznanie się z instrukcjami jednostki zmiennoprzecinkowej procesora oraz zrozumienie jak przechowywane są dane i jak działają poszczególne rejestry w koprocesorze. Dodatkowo należało przypomnieć sobie informacje związane z budową liczb zmiennoprzecinkowych m.in. długości mantysy czy wykładnika.

2. Przebieg pracy nad programem

Początkowym etapem pracy nad programem jakim należało się zająć było rozeznanie i poznanie mechanizmów w jaki sposób dokonywane są obliczenia na liczbach zmiennoprzecinkowych w komputerze.

Po przeprowadzeniu wstępnego rozeznania teoretycznego, pierwszym krokiem który wykonałem była rezerwacja miejsca na dane postaci „.float” i „.double”. Następnie zająłem się główną logiką programu, czyli operacjami na liczbach zmiennoprzecinkowych. Do implementacji operacji posłużyłem się instrukcjami oferowanymi przez język assemblera. Były to m.in. „fld” służąca do przekazywania liczby do rejestru st(0), „fst %st(n)” do przenoszenia wartości z rejestru st(0) do st(n), oraz odpowiednia instrukcja arytmetyczna jak np. „faddp” czy „fsubp”. Dla liczb zmiennoprzecinkowych podwójnej precyzji operacje te różniły się jedynie dodaniem przyrostka „l” do większości instrukcji, np. „fstl”, „fldl”. Ostatnimi rzeczami jakimi zająłem się w programie było dorobienie odpowiedniego interfejsu dla użytkownika, tak by był on w miarę możliwości funkcjonalny. Dodane przeze mnie opcje to m.in. pobieranie liczby od użytkownika, wybór trybu pracy kalkulatora, czy wyświetlanie wyników dla liczb zmiennoprzecinkowych pojedynczej precyzji.

3. Napotkane problemy

Głównymi problemami jakie napotkałem w trakcie implementacji programu były m.in. pobieranie danych od użytkownika oraz odpowiednie wyświetlanie wyników operacji przeprowadzonych na liczbach zmiennoprzecinkowych pojedynczej precyzji.

Pierwszym rozwiązanym przeze mnie problemem była kwestia pobierania danych od użytkownika. W tym celu posłużyłem się funkcją „scanf”. Działa ona w sposób podobny do funkcji „printf”, tzn. należało umieścić na stosie w kolejności adres miejsca w pamięci do której daną liczbę chcemy zapisać, adres strumienia formatującego, a następnie wywołać funkcję scanf.

Drugą z kolei rzeczą, którą należało się zająć było odpowiednie wyświetlanie wyników operacji przeprowadzonych na liczbach zmiennoprzecinkowych pojedynczej precyzji. W tym celu napisałem odpowiednią procedurę, która pobierała liczbę zmiennoprzecinkową pojedynczej precyzji, a następnie przy pomocy odpowiednich instrukcji kopiowałem i przesuwałem z niej istotne fragmenty, czyli znak liczby, wykładnik oraz mantysę. Na koniec odpowiednio sformatowaną liczbę umieszczałem na stosie, tak by została poprawnie odczytana przez funkcję printf.

4. Kluczowe fragmenty kodu

Poniżej prezentuję kluczowe procedury znajdujące się w części programów.

1. Instrukcje wprowadzania danych, tj. wybór trybu kalkulatora i wprowadzenie liczb na przykładzie liczb zmiennoprzecinkowych pojedynczej precyzji.

```
mov $SYSWRITE, %eax          # wybor trybu
mov $STDOUT, %ebx
mov $mode, %ecx
mov $mode_len, %edx
int $SYSCALL

mov $SYSREAD, %eax           # pobrane wartosci trybu
mov $STDIN, %ebx
mov $input_mode, %ecx
mov $input_mode_len, %edx
int $SYSCALL

mov input_mode, %ecx             # switch case MODE
cmp $0xa31, %ecx
je FLOAT
cmp $0xa32, %ecx
je DOUBLE
JMP QUIT

FLOAT:
mov $SYSWRITE, %eax          # podanie pierwszej liczby
mov $STDOUT, %ebx
mov $first_msg, %ecx
mov $first_msg_len, %edx
int $SYSCALL

push $first
push $input_format_float
call scanf

mov $SYSWRITE, %eax          # podanie drugiej liczby
mov $STDOUT, %ebx
mov $second_msg, %ecx
mov $second_msg_len, %edx
int $SYSCALL

push $second
push $input_format_float
call scanf

mov $SYSWRITE, %eax          # wyswietlenie trybu dodawania/mnozenia/dzielenia/odejmowania
mov $STDOUT, %ebx
mov $welcome_msg, %ecx
mov $welcome_msg_len, %edx
int $SYSCALL
```

```
        mov $SYSREAD, %eax          # wczytanie trybu od uzytkownika
        mov $STDIN, %ebx
        mov $input, %ecx
        mov $input_len, %edx
        int $SYSCALL

        mov input, %ecx             # switch case
        cmp $0xa31, %ecx
        je F_ADD
        cmp $0xa32, %ecx
        je F_SUBSTR
        cmp $0xa33, %ecx
        je F_MULT
        cmp $0xa34, %ecx
        je F_DIV
JMP QUIT
```

2. Instrukcje działań arytmetycznych na przykładzie liczb zmiennoprzecinkowych pojedynczej precyzji

```
        F_ADD:
        fld first               # zaladuj first liczbe do st(0)
        fst %st(1)              # skopiuj st(0) do st(1)
        fld second              # zaladuj second liczbe do st(0)
        faddp %st(1), %st(0 )# dodaj wartosci i zachowaj wynik w st(0)
        call SKIP
        fstp temp
        jmp QUIT
        F_SUBSTR:
        fld first
        fst %st(1)
        fld second
        fsubp %st(1), %st(0)
        call SKIP
        fstp temp
        jmp QUIT
        F_MULT:
        fld first
        fst %st(1)
        fld second
        fmulp %st(1), %st(0)
        call SKIP
        fstp temp
        jmp QUIT
        F_DIV:
        fld first
        fst %st(1)
        fld second
        fdivp %st(1), %st(0)
        call SKIP
        fstp temp
        jmp QUIT
```

3. Procedura przygotowująca liczbę zmiennoprzecinkową pojedynczej precyzji do wyświetlenia

```
PREPARE_RESULT_TO_PRINT:    # Uwaga. Floaty konwertowane sa do doubla(w wyswietlaniu printfem) -- nalezy
                            # przekonwertowac z liczby pojedynczej precyzji do podwójnej precyzji
xorl %edx, %edx
pop %edi                    # sciagniecie adresu z ostatniej funkcji tak aby pod koniec wrocic do niego -- adres powrotu z ramki stosu
popl %eax

mov temp, %eax              # ponizej kopiowanie eax do dwoch osobnych rejestrow by umozliwic "przeskanowanie" calej liczby, tj.
movl %eax, %ebx             # wyciagniecie znaku, mantysy, wykladnika.
movl %eax, %ecx
                            # przygotowanie mantysy do podwojnej precyzji
andl $8388607, %eax         # 8388607(DEC) = 0111 1111 1111 1111 1111 1111(BIN) = '1' -  mantysa w liczbie pojedynczej
                            # precyzji (23 bity) -- dzielna (to co dziele)
movl $8, %esi               # 8 (DEC) = 1000(BIN) = dlugosc -- dzielnik (przez co dziele)
divl %esi                   # podziel eax przez esi - wynik dzielenia całkowitoliczbowego - eax, reszta z dzielenia - edx.
shll $29, %edx              # 29 (DEC) = 0001 1101(BIN). Przesunięcie edx, tak by z prawej strony były same zera -
                            # wyrównanie do mantysy liczby o podwójnej precyzji.

                            #przygotowanie znaku i wykładnika do podwojnej precyzji
andl $2139095040, %ebx      # 2139095040 (DEC) = 0111 1111 1000 0000 0000 0000 0000 0000(BIN) - pozycje jedynek =
                            # pozycja wykładnika w liczbie zmiennoprzecinkowej pojedynczej precyzji
shrl $23, %ebx              # 23 (DEC) = 0001 0111(BIN). Przesunięcie pozycji, by wyciągnąć wykładnik.
addl $896, %ebx             # 896 (DEC) = 0011 1000 0000(BIN) Dodanie 896(DEC) by zapisać wykładnik na 11 pozycjach
                            # (zgodnie z podwójną precyzją).Obciążenie między formatami.
shll $20, %ebx              # 20 (DEC) = 0001 0100(BIN) -- Otrzymujemy wykładnik na liczbie 64 bitowej.

andl $-2147483646, %ecx     # -2147483646 (DEC) = 1111 1111 1111 1111 1111 1111 1111 1111 1000 0000 0000 0000 0000
                            # 0000 0000 0010(BIN) AND i OR -- operacje składające liczbę do postaci 64b.
orl %ecx, %eax
orl %ebx, %eax

pushl %eax                  # 1. czesc "double"
pushl %edx                  # 2. czesc "double"
pushl %edi                  # adres powrotu funkcji
ret
```

5. Opis uruchomienia programu

Do uruchomienia programów wykorzystałem komendy podane w bashu w postaci:

1. „gcc -m32 -ggdb calculator.s”
2. „./a.out”
3. Do debugowania „gdb a.out”

Pierwsza z komend służyła stworzeniu programu przy użyciu kompilatora języka C. „gcc” odnosi się do samego kompilatora. „-m32” określa by dane wyjściowe były zapisane w formacie 32 bitowym. „-ggdb” nakazuje użycie debuggera, natomiast „calculator.s” jest to nazwa pliku w którym znajduję się kod napisany w języku assemblera.

Druga komenda odnośni się do uruchomienia skompilowanego programu, którego domyślna nazwa to „a.out”.

Trzecia komenda odnosiła się do uruchomienia debuggera w poszukiwaniu ewentualnych błędów w trakcie tworzenia programów. Dodatkowo w trakcie debugowania programu korzystałem z takich poleceń, jak:

a. „x/gf &liczba1” – do wyświetlenia liczby, zapisanej pod danym adresem w pamięci, w systemie dziesiętnym,
b. „info register all” – do wyświetlenia zawartości wszystkich rejestrów
c. „b ” + nr linii – do ustawienia punktów przerwania debuggera
d. „stepi”, „run”,”continue” – w kolejności, do przejścia debuugera do następnej instrukcji, do uruchomienia programu oraz do przejścia do następnego punktu przerwania.

Zawartość pliku „makefile” dla programu dodającego liczby prezentuję poniżej.

```
all: calculator

adder: a.out
	gcc -m32 -ggdb calculator.s
```